Amendments to the Claims:

This listing of claims replaces all prior versions and listings of claims in the application:

Listing of Claims:

1. (Currently Amended) A method comprising:

in a processing system, receiving a rule set as a single package;

generating a dependency graph for the rule set, the dependency graph including a plurality of ranked nodes, the nodes including entity nodes, attribute nodes, condition nodes, and rule nodes, where a first attribute node points to a first condition node when a first attribute is used in a conditional expression, a first condition node points to a first rule node when a first condition associated with the first condition node is used in a first rule associated with the first rule node, and where the first rule node points to a second rule node when the first rule overrides a second rule associated with the second rule node; and

generating a sequence of processing logic for optimal processing of inputted facts according to a rank order of the nodes in the dependency graph.

2. (Original) The method of claim 1 in which processing comprises single pass execution when there are no logical loops.

3. (Previously Presented) The method of claim 1 in which processing comprises multi-pass execution when there are logical loops.

4. (Original) The method of claim 3 in which processing further comprises providing an endless loop terminating condition.

5. (Original) The method of claim 1 in which the rule set is free of logical conflicts.

6. (Original) The method of claim 1 in which generating the dependency graph comprises determining logical dependencies across rules contained in the rule set.

7. (Original) The method of claim 6 in which generating the dependency graph further comprises resolving logical conflicts using override instructions.

8. (Original) The method of claim 7 in which generating the dependency graph further comprises analyzing the rule set with a business logic generation utility optimized for one of a plurality of target programming languages and generating optimized logic for a selected target programming language.

9. (Original) The method of claim 8 in which the target programming language is Java.

10. (Original) The method of claim 8 in which the target programming language is C++.

11. (Original) The method of claim 8 in which the target programming language is Jython.

12. (Original) The method of claim 8 in which the target programming language is JavaScript.

13. (Previously Presented) The method of claim 8 in which the target programming language is Visual Basic.

14. (Original) The method of claim 8 in which the target programming language is C#.

15. (Original) The method of claim 8 in which the business logic generation utility's generated processing logic comprises a series of calls to a working memory database to retrieve, manipulate and update data.

16. (Previously Presented) A method for automating business processes comprising:

in a computer system, receiving a rule set as a single package;

determining logical conflicts within the rule set where a logical conflict exists when two or more rules receiving the same inputs result in contradictory actions;

resolving the logical conflicts; and

generating a sequence of processing logic from the rule set for optimal processing of inputted facts using the resolved logical conflicts.

17. (Original) The method of claim 16 in which resolving comprises determining override conditions in rule collision events.

18. (Original) The method of claim 16 in which generating comprises analyzing the rule set with a business logic generation utility optimized for one of a plurality of target programming languages and generating optimized business logic for the selected target programming language.

19. (Original) The method of claim 18 in which the target programming language is Java.

20. (Original) The method of claim 18 in which the target programming language is C++.

21. (Original) The method of claim 18 in which the target programming language is Jython.

22. (Original) The method of claim 18 in which the target programming language is JavaScript.

23. (Original) The method of claim 18 in which the target programming language is Visual Basic.

24. (Original) The method of claim 18 in which the target programming language is C#.

25. (Original) The method of claim 18 in which the business logic generation utility's generated processing logic comprises a series of calls to a working memory database to retrieve, manipulate and update data.

26. (Currently Amended) A computer program product, disposed on a computer readable medium, for business processing automation, the program including instructions for causing a processor to:

receive a rule set as a single package;

generate a dependency graph for the rule set, the dependency graph including a plurality of ranked nodes, the nodes including entity nodes, attribute nodes, condition nodes, and rule nodes, where a first attribute node points to a first condition node when a first attribute is used in a conditional expression, a first condition node points to a first rule node when a first condition associated with the first condition node is used in a first rule associated with the first rule node, and where the first rule node points to a second rule node when the first rule overrides a second rule associated with the second rule node; and

generate a sequence of processing logic for optimal processing of inputted facts according to a rank order of the nodes in the dependency graph.

27. (Original) The product of claim 26 in which the rule set is free of logical conflicts.

28. (Original) The product of claim 26 in which to generate the dependency graph comprises determining logical dependencies across rules contained in the rule set.

29. (Original) The product of claim 28 in which to generate the dependency graph further comprises instructions for causing the processor to:

determine logical conflicts between rules in the rule set; and

resolve the logical conflicts with override instructions.

30. (New) The method of claim 1, where node rank is used to determine an order of processing of the nodes, the nodes being ranked such that a child node has a greater value than each of its parent nodes, and where independent nodes are ranked such that a more computationally intensive node is ranked higher.